Homework 12

Charles Swarts
swarts2@illinois.edu

April 2016

1 Problem 1

1.1 question

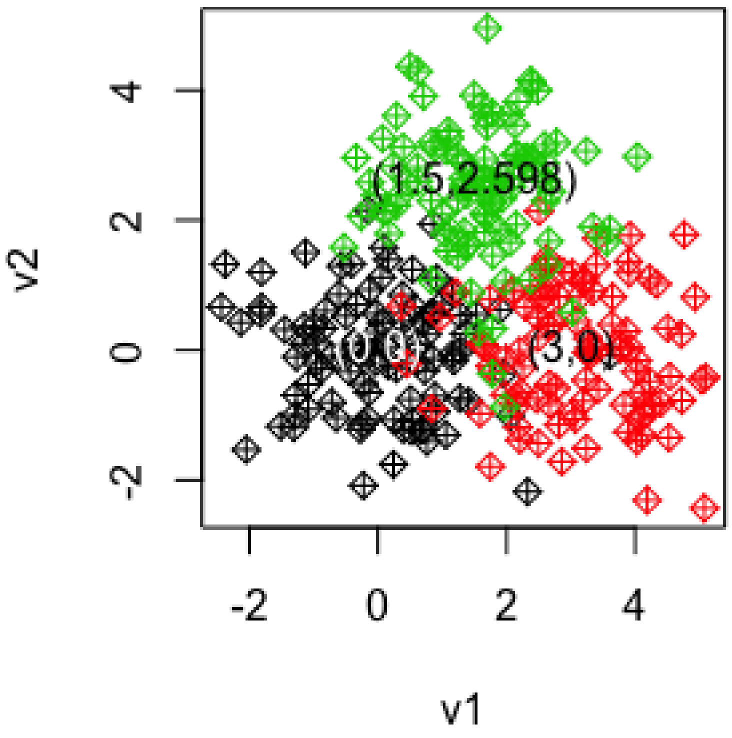
This time we're going to work on two different datasets.

Problem 1.

Generate three 2D Normal distributions with random mean vectors and unit variances. The Euclidean distance between the three random mean vectors shouldn't exceed 3. What that means is that you randomly define three 2D means, and your covariance matrices are all identity matrices. Draw 100 samples per distribution. Overall, you have 300 samples from three Gaussians. Pretend like that you don't know which sample belongs to which Gaussian.

(a) Scatterplot the samples. Use different colors or markers to distinguish samples from different distributions. Also, mark the positions of the original means. (b) Write your own code to do k-means clustering on this dataset. Show your estimated means on the scatterplot, and compare them to the original ones.

2 Answer



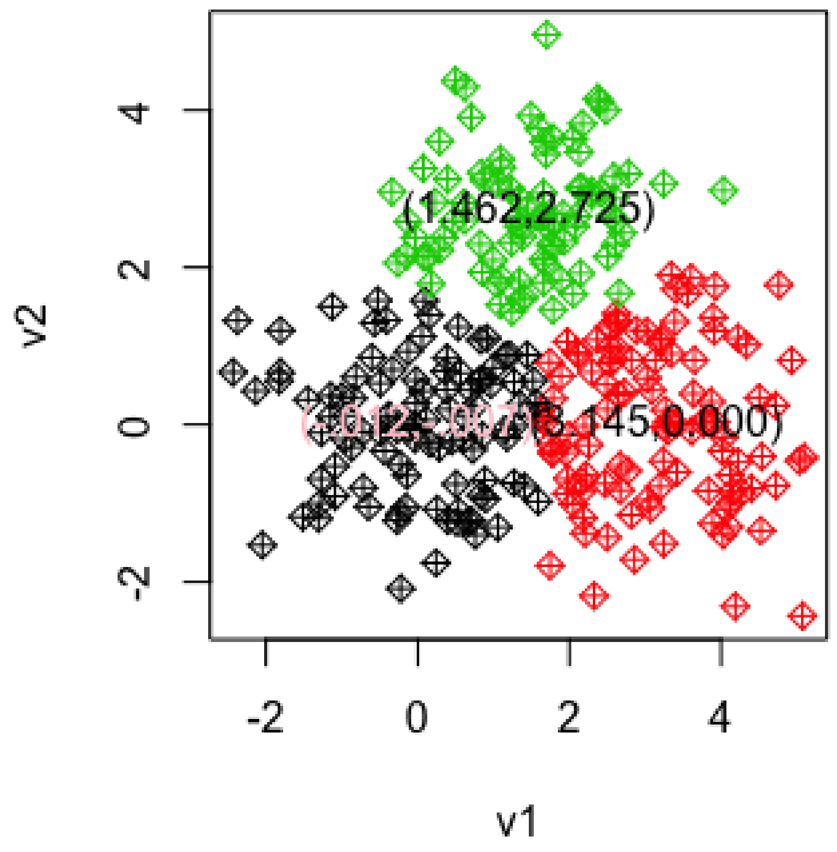
(a).

(b). My actual means were

1: $0, 0$
2: $3, 0$
3: $1.5, 2.598$
My estimated means were

1: $-0.012698883, -0.007750091$
2: $3.1458264815, 0.0007224473$
3: $1.462456, 2.725030$

They are pretty close to the mark in at least one dimension. 2 and 3 have one dimension that is off by .1.



2.1 Raw Code

```
    #For this question, I am picking the points (0,0)
#(3,0) and (1.5,2.598) each of which you will see
#have a euclidian distance of 3 from each other.

#Here the three 2D normal distributions are being
#drawn from.
v1 <- rnorm(100,0,1)
v2 <- rnorm(100,0,1)
v1 <- c(v1,rnorm(100,3,1))
v2 <- c(v2,rnorm(100,0,1))
v1 <- c(v1,rnorm(100,1.5,1))
v2 <- c(v2,rnorm(100,2.598,1))

#v3 keeps track of who's who.
```

```
v3<- factor(c(rep("x1",100),rep("x2",100),rep("x3",100)))

#Makes a dataframe to help keep track.
pointsFrame<- data.frame(cbind(v1,v2,v3))

#Here is the original plot before clustering.
plot(v2~v1,col=v3,pch=9)
text(0,0,"(0,0)",col="white")
text(3,0,"(3,0)",col=9)
text(1.5,2.598,"(1.5,2.598)",col=9)

#The points are here to keep track
#of if the algorithm has found the
#stable state.
oldPoints <- c(-1,-1,-1)

#newPoints first draws three random
#indices out of a hat.
newPoints <- sample(1:300,3,replace=FALSE)

#While not at a stable state.
while(newPoints[1]!=oldPoints[1]|newPoints[2]!=oldPoints[2]|newPoints[3]!=oldPoints[3])
{
  #This is how I keep track of the old points.
  oldPoints <- newPoints
  #Here is the space for the three clusters.
  clusterOne <- c()
  clusterTwo <- c()
  clusterThree <- c()

  #This sorts all threehundred points by cluster.
  for(i in 1:300)
  {
    #First the square distance from the k points
    #is calculated for every point in the field.
    SqrdistanceFrom1 <- (pointsFrame[i,1]-pointsFrame[oldPoints[1],1])^2
    SqrdistanceFrom1 <- SqrdistanceFrom1+(pointsFrame[i,2]-pointsFrame[oldPoints[1],2])^2
    SqrdistanceFrom2 <- (pointsFrame[i,1]-pointsFrame[oldPoints[2],1])^2
    SqrdistanceFrom2 <- SqrdistanceFrom2+(pointsFrame[i,2]-pointsFrame[oldPoints[2],2])^2
    SqrdistanceFrom3 <- (pointsFrame[i,1]-pointsFrame[oldPoints[3],1])^2
    SqrdistanceFrom3 <- SqrdistanceFrom3+(pointsFrame[i,2]-pointsFrame[oldPoints[3],2])^2

    #These if statements deposit the points in the
    #appropriate cluster.
    if(SqrdistanceFrom1<=SqrdistanceFrom2&SqrdistanceFrom1<=SqrdistanceFrom3)
    {
      clusterOne <- c(clusterOne,i)
    }
    else if(SqrdistanceFrom2<=SqrdistanceFrom3&SqrdistanceFrom2<=SqrdistanceFrom1)
    {
      clusterTwo <- c(clusterTwo,i)
    }
    else if(SqrdistanceFrom3<SqrdistanceFrom2&SqrdistanceFrom3<SqrdistanceFrom1)
    {
      clusterThree <- c(clusterThree,i)
    }
    #This warns me of dirty bugs.
    else
    {
      print("bug")
    }
  }

  #Next the means of the clusters are calculated.
  clusterOneMean <- mean(pointsFrame[clusterOne,1])
  clusterOneMean <- c(clusterOneMean,mean(pointsFrame[clusterOne,2]))

  clusterTwoMean <- mean(pointsFrame[clusterTwo,1])
  clusterTwoMean <- c(clusterTwoMean,mean(pointsFrame[clusterTwo,2]))

  clusterThreeMean <- mean(pointsFrame[clusterThree,1])
  clusterThreeMean <- c(clusterThreeMean,mean(pointsFrame[clusterThree,2]))

  #Finally the point in each cluster closest to the
  #mean is made to be the new k point for the cluster.
  newPoints[1]<- clusterOne[which.min((pointsFrame[clusterOne,1]-clusterOneMean[1])^2+
                              (pointsFrame[clusterOne,2]-clusterOneMean[2])^2)]
  newPoints[2]<- clusterTwo[which.min((pointsFrame[clusterTwo,1]-clusterTwoMean[1])^2+
                              (pointsFrame[clusterTwo,2]-clusterTwoMean[2])^2)]
  newPoints[3]<- clusterThree[which.min((pointsFrame[clusterThree,1]-clusterThreeMean[1])^2+
                              (pointsFrame[clusterThree,2]-clusterThreeMean[2])^2)]
  #Next the algorithm will reevaluate if it is in a stable state.
  #If not, it will continue.
}

#This is space for the new classifications.
clustersAfter <- rep(" ",300)

#Now the labels are reapplied.
#The awkward pairing makes sure the coloring lines up.
clustersAfter[clusterOne] <- "x2"
clustersAfter[clusterTwo] <- "x3"
clustersAfter[clusterThree] <- "x1"

#This factorized clustersAfter to make sure
#the clustering "sticks."
clustersAfter <- factor(clustersAfter,levels=c("x1","x2","x3"))

#And here is the plot of the data afterwards.
plot(v2~v1,col=clustersAfter,pch=9)
text(-0.012,-0.007,"(-.012,-.007)",col="pink")
text(3.1458,0,"(3.145,0.000)",col=9)
text(1.462,2.72,"(1.462,2.725)",col=9)
```

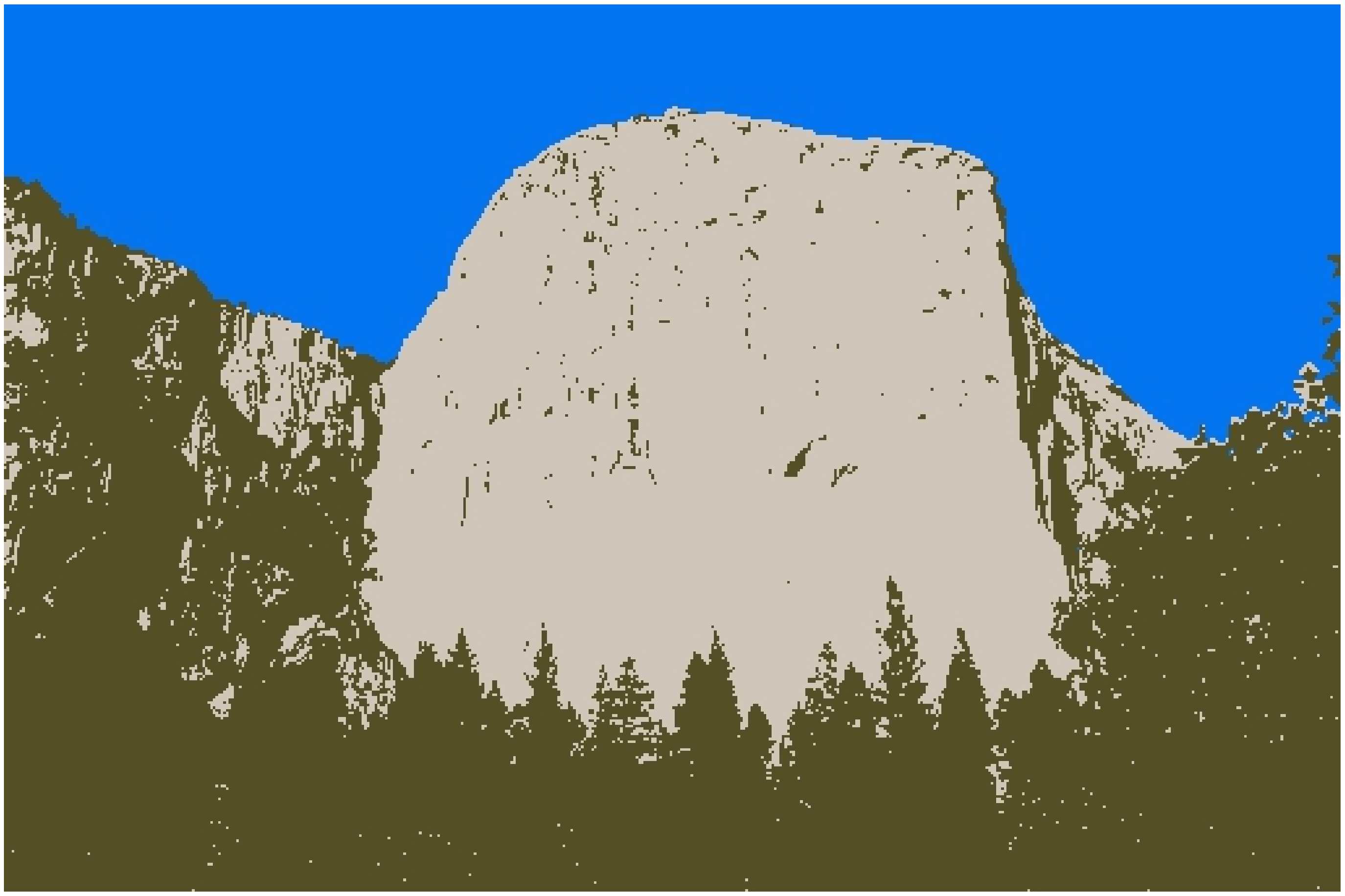
3 Problem 2

3.1 question

Problem 2.

Attached is a picture of El Capitan in the Yosemite National Park. Do the k-means clustering using your code to cluster the pixels into three groups: ”sky”, ”rock”, and ”tree”. Now, redraw the picture by using your three means you found (which means that every pixel can have to choose its color from the three mean values). Note that if you ignore all the positions of the pixels, you can reformulate the image into a matrix of (3 X of all pixels).

3.2 answer



Here is the finished product. It is a clusterized, or as I like to call it, ”Warhol-ized” photo of the original. And I diffed it with the one on Piazza, they are different despite looking very similar. I am assuming that because the question did not say to label the photo, I don’t have to. Anyways, the real story with this question is in the code. My code is generalized to any number of colors you wish to have, and it prevents infinite loops. Feel free to use it actually... if you want.

raw code

```
#This library is necessary to read and write jpegs
```

```
install.packages("jpeg")
library(jpeg)

#Read in the jpeg
myjpg <- readJPEG("/users/CharlesBSwarts/Desktop/elcapitan3.jpg",native=FALSE)

#Set the number of colors you want it to have
k <- 3

#Seperate the 3D array into 3 1D vectors.
RED <- as.vector(myjpg[,,1])
GREEN <- as.vector(myjpg[,,2])
BLUE <- as.vector(myjpg[,,3])

#Ponts is a vector of indices that are the mean/
#starter points of the next iteration of the
#k-means clustering.
newPonts <- c()
length(newPonts) <- k
oldPonts <- rep(-1,k)

#Need to pick k random points.
newPonts <- sample(1:(dim(myjpg)[1]*dim(myjpg)[2]),k,replace=FALSE)

#Now, clusters are stored in a jagged array,
#but R does not support jagged arrays except
#in lists of lists. So this clus is going to
#be the list that holds the other lists.
clus <- list()
length(clus) <- k

#DisSqr is the square distance of each point
#from the means.
DisSqr <- as.data.frame(matrix(ncol=k,nrow=dim(myjpg)[1]*dim(myjpg)[2]))

#clusMean is the mean of the values for each
#of the colors
clusMean <- as.data.frame(matrix(ncol=k,nrow=3))

#This stopper is here just in case we get an
#overly noisy photo or for some reason the
#algorithm begins to cycle.
stopper <- 0

#If the old points equal the new points, then
#we have found a minimum.
while(!all(oldPonts==newPonts)&stopper<100)
{
  #Store the new points as the old points.
  oldPonts <- newPonts
  #reset the clus list.
  clus <- list()
  #This calculates the square distance of every
  #point from every k point.
  for(i in 1:k)
  {
    DisSqr[,i] <- (RED-RED[oldPonts[i]])^2+(GREEN-GREEN[oldPonts[i]])^2+
                                        (BLUE-BLUE[oldPonts[i]])^2
  }

  #This calculates the minimum in every row
  values<- apply(DisSqr,1,min)

  #This sorts the respective points to their
  #respective cluster.
  for(i in 1:k)
  {
    clus[i] <- list(which(DisSqr[,i]==values))
  }

  #This calculates the mean value for each cluster
  for(i in 1:k)
  {
    clusMean[,i] <- c(mean(RED[unlist(clus[i])]),mean(GREEN[unlist(clus[i])]),mean(BLUE[unlist
  }

  #This chooses new starting k points based
  #on the mean value of each cluster.
  for(i in 1:k)
  {
    newPonts[i]<- unlist(clus[i])[which.min((RED[unlist(clus[i])]-clusMean[1,i])^2+(GREEN[unlist
  }
  #Increment the nasty stopper to avoid
  #infinite loops
  stopper <- stopper+1
}



#This creates vectors to be reinverted back
#into an image.
mapRED <- c()
mapGREEN <- c()
mapBLUE <- c()
length(mapRED) <- dim(myjpg)[1]*dim(myjpg)[2]
length(mapGREEN) <- dim(myjpg)[1]*dim(myjpg)[2]
length(mapBLUE) <- dim(myjpg)[1]*dim(myjpg)[2]


#This function puts the k point's color
#onto every pixel in its cluster.
for(i in 1:k)
{
```

```
mapRED[unlist(clus[i])] <- RED[newPonts[i]]
mapGREEN[unlist(clus[i])] <- GREEN[newPonts[i]]
mapBLUE[unlist(clus[i])] <- BLUE[newPonts[i]]
}


#This creates the final array for the writeJPEG
#function
imgArr <- array(dim=c(dim(myjpg)[1],dim(myjpg)[2],3))
imgArr[,,1] <- mapRED
imgArr[,,2] <- mapGREEN
imgArr[,,3] <- mapBLUE

#And this outputs the photo.
writeJPEG(imgArr,target="/users/CharlesBSwarts/Desktop/Warhol.jpg",quality=1)
```